Dropout and Scaling in Neural Networks

1 Introduction

Dropout is a regularization technique used in neural networks to prevent overfitting. During training, with a dropout probability p, each neuron (or unit) in a layer is randomly set to zero with probability p and retained with probability $1 - p$.

2 Mathematical Justification

2.1 During Training

Mathematically, if we denote the output of a layer as y and the weights as w, then during training, the output can be represented as:

$$y_{\text{train}} = \text{Dropout}(w \cdot x) \cdot m$$

where m is a mask vector with elements drawn from a Bernoulli distribution:

$$m_i = \begin{cases} 1 & \text{with probability } (1-p) \\ 0 & \text{with probability } p \end{cases}$$

This effectively scales down the activations because only a fraction $(1 - p)$ of the neurons are active.

2.2 Expected Value Calculation

The expected output during training is:

$$E[y_{\text{train}}] = E[m] \cdot w \cdot x = (1-p) \cdot w \cdot x$$

This means that the expected output during training is scaled by a factor of $(1 - p)$.

2.3 During Inference

During inference, dropout is not applied; hence, all neurons are active. To maintain consistency with the training phase, the outputs must be scaled by the same factor used during training:

$$y_{\text{inference}} = w \cdot x \cdot (1 - p)$$

2.4 Scaling

To address the change in input distribution due to dropout, we can scale the outputs during inference. If we denote the output of the layer as y_{raw} without any dropout, we adjust for the expected scaling during inference:

$$y_{\text{scaled}} = y_{\text{raw}} \cdot (1 - p)$$

Alternatively, during training, to counterbalance the scaling effect of dropout, we can scale the non-zero activations by:

$$y_{\text{scaled}} = y_{\text{train}} \cdot \frac{1}{1 - p}$$

3 Conclusion

By applying dropout during training and appropriately scaling the outputs during inference, we can ensure that the model maintains consistency in its expected behavior across both phases. This approach helps mitigate overfitting while preserving the model's ability to make accurate predictions when deployed.